Evaluating the difficulty for novice engineers in learning and using Transition Systems for modeling software systems

Mrityunjay Kumar
IIIT Hyderabad
India
mrityunjay.k@research.iiit.ac.in

Venkatesh Choppella
IIIT Hyderabad
India
venkatesh.choppella@iiit.ac.in

ABSTRACT

Modern software products are complex systems and are better comprehended when engineers can think of the software as a system. Systems Science suggests that learning about a complex system is aided by modeling. It stands to reason that if we can help novice engineers model the software products as systems, it should improve their comprehension.

One way of learning through modeling is to use Transition Systems to build models that we proposed in a previous paper. This requires the engineers to learn the vocabulary of Transition Systems and a way to use it to model software systems. The question arises: is it difficult to learn and use transition systems vocabulary? We hypothesize that it is not, because its vocabulary is small, and it builds on the concepts learned in other courses like Theory of Computation and Discrete Mathematics - finite-state machines and set theory.

To test this hypothesis, we designed a short intervention (one lecture and one project) in a software engineering course for two cohorts of students from two different environments. We taught them basic concepts of Transition Systems and how systems can be modelled using its vocabulary and evaluated their *performance* on a modeling project. We also administered a survey to evaluate their *perception* of the topic.

Both the cohorts scored well on the project and reported agreement with ease of learning and use of Transition Systems when surveyed. Based on the knowledge demonstrated and the survey feedback, we conclude that it is not difficult for them to learn the vocabulary of Transition Systems and its use. This result gives confidence to start designing longer intervention to promote use of systems modeling and study their effectiveness with large software systems.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**.

KEYWORDS

transition systems, system modeling, novice engineers

COMPUTE '23, December 09–11, 2023, Hyderabad, India

ACM ISBN 979-8-4007-0840-4/23/12…$15.00
https://doi.org/10.1145/3627217.3627223

ACM Reference Format:
Mrityunjay Kumar and Venkatesh Choppella. 2023. Evaluating the difficulty for novice engineers in learning and using Transition Systems for modeling software systems. In *16th Annual ACM India Compute Conference (COMPUTE '23), December 09–11, 2023, Hyderabad, India.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3627217.3627223

1 INTRODUCTION

The move to cloud for software products is accelerating and SaaS (Software-as-a-Service) is becoming an established way of building and delivering software products. A typical SaaS product is a distributed, multi-tenant system using multiple external APIs and a service-oriented architecture. This is changing the way software is built [16] and is also increasing the complexity of the software. It is also making it more difficult for the novice engineers to comprehend the products sufficiently in the limited time allotted for onboarding. In [9], we proposed that using Transition Systems to model software can be a viable way for novice engineers to design and understand systems. However, this requires the novice engineers to learn and be able to use Transition Systems vocabulary.

How difficult is it to learn and use the Transition Systems for modeling systems? Answer to this question impacts how well and how fast it can be adopted by the students and instructors when introduced in the curriculum and so it is important to answer this.

We framed the following research question (RQ) and the null hypothesis (H0):

RQ: How difficult is it for undergraduate computer science students to learn and use Transition Systems vocabulary?

Our experiment consisted of doing a short intervention to two cohorts of students doing Software Engineering course at two different undergraduate colleges. The hypothesis was that if students can learn with such a short intervention, it is not difficult for them to learn and use.

Section 2 lays out related work on applying systems concepts to software systems and using Transition Systems for modeling. Section 3 defines Transition Systems and provides an example to illustrate what we mean by modeling. Section 4 describes the experiment design and the rubric for evaluation. Section 5 discusses the results and shows the analysis of the results obtained from the experiment. Finally, Section 6 lays out the conclusion, including threats to validity and future work.

2 RELATED WORK

As software systems get complex, applying the systems approach to software can be of value.

Von Bertalanffy [17] defined a system as a "*set of elements standing in interrelation among themselves and with the environment*"

and a dynamical system as a set of state variables whose changes are represented typically by simultaneous, first order differential equations. Hmelo-Silver and Azevedo [7] suggest that "*in complex systems, aggregate nature of the system is not predictable from the isolated components but occurs through the interaction of multiple components (for example, human body)*" and conclude that "*..It [study of complex systems] is hard because learning about these systems challenges cognitive, metacognitive, and social resources.*" These descriptions and challenges apply to modern software systems well (though with a discrete system dynamics).

Models play an important role in learning about systems. Sterman [15] suggests that information feedback about the real world alters our mental models in the process of learning and that is required to learn about complex systems. Morecroft [13] talks of a model as a "*tangible aid to imagination and learning, a transitional object to help people make better sense of a partly understood world*". Hashem and Mioduser [6] suggest that students can learn better if they model the complex system. In CS, while models do get used, they either get used to formalize the requirements and design, or to automate the generation of code from models. Using them for SaaS (Software as a Service) software products to aid learning and comprehension is uncommon. We propose that we should use system modeling to aid system comprehension.

Software systems are discrete-time dynamical systems which can be modelled well as Transition Systems [1]. Zeigler presented DEVS (Discrete Events System Specification) as a seven-tuple [18] to capture the dynamics of discrete events system. Moore [12] and Mealy [10] machines are finite-state machine representations along similar lines. We have used a similar notation to define iterative and interactive transition systems for representing algorithmic systems [2] as well as software systems [9]. In this paper, we use the same notation when we teach modeling to the Software Engineering students. While UML is a prevalent modeling language, its focus on diagrams and a large vocabulary makes it a more difficult skill to master [5, 14].

Transition Systems uses state machine concepts and leverages set theory for its notation, both of which are part of CS curriculum in early years of undergraduate studies. Constructivist theory argues that learners construct their knowledge themselves using external inputs [3] and this is aided when the learner engages with the material and can connect it with their existing knowledge [4, 11]. That is why we believe that students should find it easy to learn and use Transition Systems concept, it connects well with what they already know.

3 TRANSITION SYSTEMS

A system can be modeled using a Transition Systems formalism [1, 18] and this formalism can be extended to software systems [2].

3.1 Transition Systems Definition

A **Transition System** for a software system is a six-tuple

$\{X, X^0, U, f, Y, h\}$ where X is the set of states, X^0 is the set of initial states, U is the set of actions, f is the transition function which is a subset of $(X \times U) \times X$, Y is the set of Observables (or output space), and h is a *display map*, mapping states to Observables, $y = h(x)$.

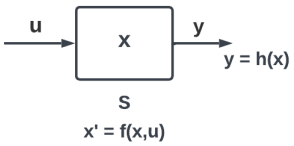


Figure 1: Transition System S (x: state, u: action, y: observable, h: display map, f: transition relation)

A Transition System models the dynamics of the software system (see Fig 1).

When applying this to a software system, it should be noted that an action ($u \in U$) may include data as well. For example, an action "read" for a database system may supply table and column names to read from. We assume type of U is defined appropriately.

3.2 Transition Systems Example

We will take a simple system and model it using the Transition Systems vocabulary. Like button on LinkedIn is a well-used functionality, we will model it here.

Here is a summary of the Like button behavior. Hover (mouse interface) or longpress (touch interface) shows a list of icons to choose from, irrespective of whether current state is liked or not liked. Selecting one of the icons sets the state to that icon. Single click (or tap) on an already selected icon marks it as not liked while a single click (or tap) when nothing has been previously selected will select the default icon (thumbs up).

Transition System Model: For the six tuple $\{X, X^0, U, f, Y, h\}$ for this system,

$$
\begin{aligned}
X &= \{Selected(iconname), Selecting\} \\
X^0 &= \{NotLiked\} \\
U &= \{hover/longpress, click(iconname)\} \\
Y &= \{icon - name, iconlist\} \\
iconlist &= \{thumbwhite, thumbblue, clap, heart, smile\}
\end{aligned}
$$

The transition function can be written as follows:

$$
\begin{aligned}
f(x, hover/longpress) &= Selecting \\
f(x, click(i)) &= Selected(i)
\end{aligned}
$$

The display map can be written as follows:

$$
\begin{aligned}
h(Selected(iconname) &= iconname \\
h(Selecting) &= iconlist
\end{aligned}
$$

As can be seen, the behavior is succinctly captured in a few equations of the transition function and display map. Writing this using Transition Systems vocabulary makes the components of the system more understandable, and this can be used for multiple purposes, including generating test cases, identifying missing scenarios and validating the behavior against the specification.

4 DESIGN OF EXPERIMENT

We designed an experiment to intervene in software engineering course classrooms of two different universities in India. We use the data from the project evaluations and surveys from these two cohorts to test the hypothesis. We use both *performance* and *perception* to check difficulty. We consider the topic to be difficult if a) *performance* is poor (mean total score for the students is less than 50%), and 2) *perception* is poor (less than 50% students disagree with the ease of learn and use questions in a survey).

4.1 Intervention

We designed one lecture worth of material for each cohort that was delivered as part of the regular course schedule. Students were provided with additional lecture notes for review purposes, and a project that they had to submit in two weeks. The content was tailored to fit the Software Engineering course they were taking and their current knowledge of CS topics.

Lecture content:[1] In addition to an industry perspective and needs from a new campus hire (to motivate the learning), the lecture covered these points: engineering systems and their dynamics (spring-mass system, pendulum, etc.), structural and behavioral models, physical models to computational models, discrete dynamics using state machines and diagrams, transition systems vocabulary of six-tuple [9], modeling simple systems (light bulb, like feature on social media, etc.). Since Cohort B consisted of students much ahead in their college journey., we briefly covered dynamics of large systems (which consist of multiple component systems) for them using examples of larger systems.

A few weeks later after the lecture and project submissions, they were asked to complete a survey about this topic.

4.2 Cohorts

Cohort A (N=12) consisted of undergraduate second year CS students from a leading liberal arts college. These students had completed basic programming, data structure and algorithms course before this course. However, they did not have any experience in building systems, not even simple web applications. Nine students submitted the project. Eleven responded to the survey.

The intervention for Cohort A consisted of one lecture of 120 minutes which was delivered online, including a class activity.

Additional reading material was shared with them after the class to illustrate the concept through more examples and description. They were then given a project that consisted of two parts:

(1) Model a simple UI system (random walk on a grid) using transition system vocabulary. This required them to understand the actions and observables and apply the definition of a transition system.
(2) Model a tic-tac-toe playing system (that plays with a human player) using transition system vocabulary. This required them to think about how the software will play the game and manage the board, and then model these dynamics through transition system vocabulary.

They worked individually on the project.

[1]Please reach out to the first author for more details on the contents of lecture, reading material, and project

Cohort B (N=145, Group size=5) consisted of undergraduate third-year and fourth-year students, as well as Masters students at a top-tier CS university. The undergraduate and masters students ratio was 60:40. All the students had completed most of their required courses. They had experience in building systems. Twenty-seven groups submitted the project. One hundred one individuals responded to the survey.

The intervention for Cohort B consisted of one lecture of 90 minutes which was delivered in-person. Additional reading material was shared with them after the class which was of a similar nature to the one for Cohort A, but focused on larger systems on which their project was based. The project asked them to analyze three aspects of an online appointment booking system and write their transition systems model.

These students worked in groups of five on their project due to the way the class was structured for other assignments and projects.

4.3 Project Evaluation

The project was graded for the course requirements. For the purpose of this paper, the relevant parts of their submissions were re-evaluated for demonstrated competencies around identification and representation of a transition system model and its components, using the rubric in Table 1 which has five criteria (dimensions) and three scoring levels. The criteria correspond to aspects of the transition system definition.

4.4 Survey design

The survey given to the cohorts asked two rating questions (in addition to other course-related questions):

(1) I found transition systems approach to modeling easy to learn (for Cohort B, this was phrased as "I found it easy to learn TSML^2")
(2) I found transition systems approach to modeling easy to use (for Cohort B, this was phrased as "I found it easy to build a model using TSML^2")

Students had to rate them on a scale of 1 (Strongly Disagree) to 5 (Strongly Agree).

The questions were framed as "easy to learn/use..." (as opposed to "difficult to learn/use...") to keep the tone positive. We report the % of respondents who disagreed (score 1 and 2) with these rating questions, which means they didn't agree that transition systems is easy to learn or use.

5 ANALYSIS

We analyzed the scores from the project submission evaluation (Table 2) and survey data (Table 3 and Figure 2).

Project submissions were graded using the rubric in Table 1, each criterion was scored between 0 and 20, and overall score was obtained by adding individual criteria scores. Survey responses were scored as the percentage of students who responded 1 (Strongly disagree) and 2 (Disagree) for each of the two questions around ease of learning and use.

Table 1: Project Rubric

	Meets expectations	Approaching expectations	Does not meet expectations
R1: State space (is adequate)	Identifies key information to be stored in state and can write them in a set theoretic notation	Identifies some information of the state, uses a mix of set theoretic	Doesn't identify any aspect of the state
R2: Action space	Identifies the key actions on the system that trigger a behavior	Identifies some of the actions	Identifies little or no actions
R3: Transition Function	Captures the behavior as a function of current state and current action and covers adequately, using set theoretic notations	Covers only a part of the behavior of the system, doesn't use set theoretic notation adequately	Covers little or no behavior
R4: Observable and Display map	Defines the observable space adequately to match the system output and defines display map keeping in mind the state space, uses set theoretic notations well	Covers only a part of the system output through observables, doesn't use set theoretic notation adequately	Misses observables or display map or covers very little
R5: System Definition	A good model for the system, even if it is at a different level of abstraction than desired	The model is only partly reflective of the system	The model does not reflect the system at any abstraction level

Table 2: Project scores for Cohort A and Cohort B. 9 submissions from Cohort A, 27 submissions from Cohort B

(a) Means across cohorts

Cohorts	Cohort A	Cohort B
R1: State space	19.4	18.1
R2: Action space	20	19.6
R3: Transition function	10.6	15.2
R4: Observable and Display map	11.1	14.8
R5: System Definition	15.6	17
Total Score	76.7	84.8

(b) Summary stats of individual rubric items for Cohort A

Metric	Median	Mean	Min	Max
R1: State space	20	19.4	15	20
R2: Action space	20	20	20	20
R3: Transition function	10	10.6	0	15
R4: Observable and Display map	10	11.1	5	20
R5: System Definition	15	15.6	10	20
Total Score	75	76.7	60	90

(c) Summary stats of total score across the cohorts

Metric	Median	Mean	Min	Max
Cohort A	75	76.7	60	90
Cohort B	90	84.8	50	100

(d) Summary stats of individual rubric items for Cohort B

Metric	Median	Mean	Min	Max
R1: State space	20	18.1	10	20
R2: Action space	20	19.6	10	20
R3: Transition function	20	15.2	10	20
R4: Observable and Display map	10	14.8	10	20
R5: System Definition	20	17	10	20
Total Score	90	84.8	50	100

5.1 Project Submission analysis

Tables 2a and 2c show the statistics for both the cohorts. First point to note is that students did well overall, with mean total scores being 76.7 and 84.8 respectively for Cohort A and B. Both cohorts did worse on R3 (Transition function) and R4 (Observables and Display map) compared to other criteria. Cohort B was better on all criteria in general, but they too were worst in R3 and R4.

However, the highlight here is that Cohort A did surprisingly well. Cohort A had only a few programming and data structures courses completed before their software engineering course, and they worked on their project alone. When we compare this with Cohort B, which had many more CS courses completed and had knowledge of system development, Cohort A comparative performance is remarkable. This suggests that the subject matter is easy enough that first year course content is enough for it. If so, we can introduce the content earlier in the curriculum rather than doing it in a software engineering course and give more time to absorb and apply this concept.

5.1.1 Cohort A performance. Table 2b shows the summary statistics for Cohort A. Total score (last row) suggests good performance overall by these students (median 75, mean 76.7, min 60, max 90). Students had the worst performance in R3 (Transition function) criteria (median 10, mean 10.6, min 0) closely followed by R4 (Observable and Display map) criteria (median 10, mean 11.1, min 5). Transition functions are the key to the dynamics of the system,

[2]Transition Systems Modeling Language

students struggled in capturing the dynamics in a notationally coherent manner even when they could describe them reasonably well in language. Poor performance on R4 (Observable and Display map) is surprising. Most of the issues students had in this criterion was in defining the Display map as a function of State X and ensuring it matches the Observable (Y). This may be due to their lack of understanding of sets and functions and how to define them well. Lack of clarity of functions is also apparent in the low score on R3 (Transition functions). In some cases, students got confused between f and h and mixed their definitions. This points to conceptual understanding gap which can be remedied with more time spent in the class. Given good performance on Total Score, (mean 15.6, median 15, min 10), it seems they were able to comprehend what goes in a system definition reasonably well.

5.1.2 Cohort B performance. Table 2d shows the summary statistics of Cohort B. Total score (last row) suggests very good performance overall (median 90, mean 84.8, min 50, max 100). Students had worst performance in R4 (Observable ad Display map) criteria (median 10, mean 14.8, min 10). Many student groups didn't write h at all, while others didn't write it completely. This cohort did much better on R3 (Transition function) compared to Cohort A; even though the mean is only 15.2, median is 20. However, notation was still a problem for many. Given that this was a larger system, a few students also struggled in defining the state space and action space correctly for the given system and scored lower than Cohort A on R1 (State space) and R2 (Action space). Some of the students also wrote all the spaces and functions in a programmatic and object-oriented idiom, instead of mathematical notation, thus making it verbose. Given good performance on Total Score, (mean 17, median 20, min 10) it seems they were able to comprehend what goes in a system definition reasonably well.

5.2 Survey response analysis

Table 3 shows % of students who rated the questions from 1 (Strongly disagree) to 5 (Strongly agree). For question "I found transition systems approach to modeling easy to learn.", we see that none of the Cohort A students disagreed (rated 1 or 2), for a total of 0%, while we see a higher disagree %, a total of 21% (5% rated 1, 16% rated 2).

For "I found transition systems approach to modeling easy to use", we see a higher % of disagreement from Cohort A than for the previous question: none disagreed strongly (rated 1), and 9% disagreed (rated 2), for a total of 9% disagree. For Cohort B too, much higher % was seen compared to previous question: 9% disagreed strongly (rated 1), while 22% disagreed (rated 2), for a total of 31%.

There are two things that stand out:

(1) ease of use has more disagreement than ease of learn.
(2) Cohort B has much higher disagreement % for both questions compared to Cohort A.

For the first point, higher disagree % for ease of use is expected because using is a higher order skill (Revised Bloom's taxonomy [8]) and is likely to require more time than one lecture.

Second point is very interesting. We can think of two reasons. One reason could be that while Cohort A was not exposed to any other modeling language (like UML) during the course, Cohort B was taught UML (Unified Modeling Language), and also had many students (about a third) who had prior knowledge of UML and it is possible that Cohort B compared with their UML knowledge and effort in mind when responding. However, given the question was clear enough, this seems unlikely. Another reason could be that the knowledge acquired so far (which is quite a lot) by these students actually makes it difficult for them to wrap their head around the notion of Transition Systems while Cohort A has not covered much ground in CS to have an opinion either way and fared better. We do not have enough justification for the difference, but this is an interesting aspect to explore: Do less experienced students find it easier than more experienced ones?

Table 3: Survey Responses from cohorts on Likert scale (1 - Strongly disagree, 5 - Strongly agree)

Question	1	2	3	4	5
Cohort A (Easy to learn)	0%	0%	36%	36%	27%
Cohort A (Easy to use)	0%	9%	36%	45%	9%
Cohort B (Easy to learn)	5%	16%	35%	38%	7%
Cohort B (Easy to use)	9%	22%	40%	28%	2%

6 CONCLUSION

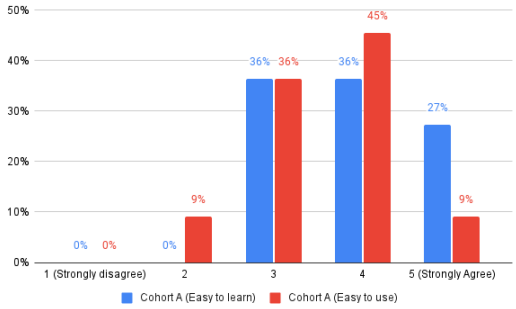
Modeling software systems as transition systems can improve system comprehension for novice engineers and graduating students. We wanted to understand whether it is difficult to learn and use transition systems approach for modeling systems. Our hypothesis is that it is not difficult to do so. To test the hypothesis, we designed a short intervention for Software Engineering course at two colleges. This consisted of one lecture delivered as part of regular course schedule, a project to test their learning, and a survey to get their feedback. We designed a rubric and assigned a score between 0 and 100 to each of the submissions.

The analysis of project evaluation scores suggests that the students do not find Transition Systems difficult to learn or use. Mean score for each cohort was more than 75%. Cohort A performed well too, which was surprising, given that they had much less experience of CS courses (since they were in their year two of undergraduate course). This points to the fact that deep background in CS courses may not be required to learn and use Transition Systems.

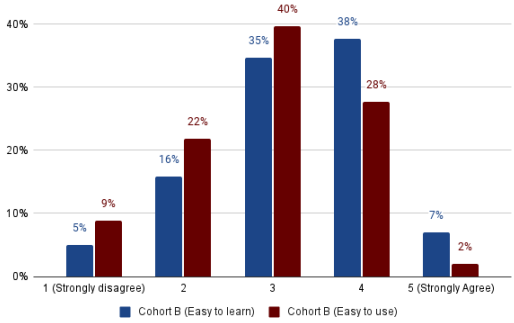
The survey responses are surprising too. Cohort B had larger percentage of students reporting that it was difficult to learn and use Transition Systems, compared to Cohort A. This is in spite of the fact that Cohort B has good CS background. One hypothesis is that maybe more experience with CS courses makes them unwilling or unable to absorb the Transition Systems concept. This needs to be investigated further.

Threats to validity. It is important to note a few aspects of this experiment that may impact the validity of the results: a) The two cohorts chosen for the experiment were quite different from each other, and hence the comparison results may not be useful, b) One of the cohorts may be considered too small to produce meaningful insights from the survey data (which yielded a surprising comparative result), c) One lecture may be too small an intervention to meaningfully conclude from the results, d) All evaluations were done by single evaluator (author), therefore the results may have suffered from a bias, even though all efforts were made to avoid it.

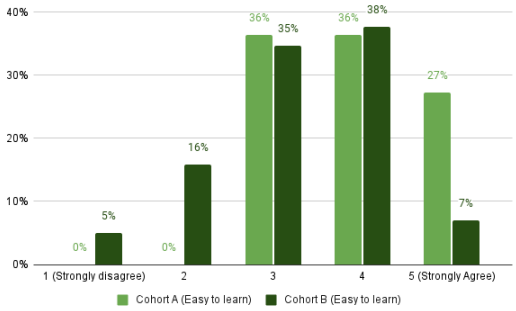
Figure 2: Survey responses comparison. 11 respondents from Cohort A, 101 respondents from Cohort B



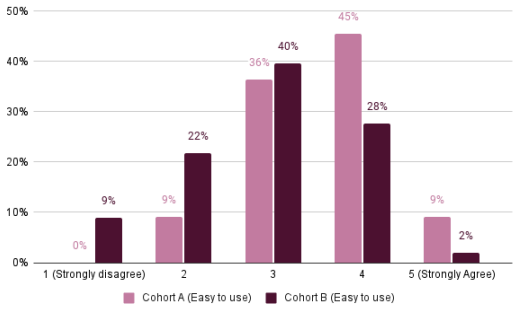
(a) Cohort A responses



(b) Cohort B responses



(c) Comparison of ease-of-learning question across cohorts



(d) Comparison of ease-of-use question across cohorts

Future work. Our future work will be focused on these aspects: a) Design and test large interventions (workshops and half-semester courses) to teach Transition Systems concepts for modeling software systems, b) Perform think-aloud interviews and other qualitative evaluation of how students use Transition Systems concepts, c) Evaluate how this modeling aids in system comprehension, d) Teach Transition Systems concept and modeling in courses earlier in CS education and outside Software Engineering courses.

REFERENCES

[1] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. 2017. *Transition Systems.* Springer International Publishing, Cham, 3–25. https://doi.org/10.1007/978-3-319-50763-7_1

[2] Venkatesh Choppella, Kasturi Viswanath, and Mrityunjay Kumar. 2021. Algodynamics: Algorithms as systems. In *2021 IEEE Frontiers in Education Conference (FIE).* IEEE, 1–9.

[3] Rosalind Driver, Hilary Asoko, John Leach, Philip Scott, and Eduardo Mortimer. 1994. Constructing scientific knowledge in the classroom. *Educational researcher* 23, 7 (1994), 5–12.

[4] Robert M Gagne. 1970. *The conditions of learning.* Holt, Rinehart and Winston,.

[5] Martin Grossman, Jay E. Aronson, and Richard V. McCarthy. 2005. Does UML Make the Grade? Insights from the Software Development Community. *Information and Software Technology* 47, 6 (2005), 383–397.

[6] Kamel Hashem and David Mioduser. 2013. Learning by Modeling (LbM): Understanding Complex Systems by Articulating Structures, Behaviors, and Functions. *International Journal of Advanced Computer Science and Applications* 4, 4 (2013). https://doi.org/10.14569/IJACSA.2013.040414

[7] Cindy E. Hmelo-Silver and Roger Azevedo. 2006. Understanding Complex Systems: Some Core Challenges. *Journal of the Learning Sciences* 15, 1 (Jan. 2006), 53–61. https://doi.org/10.1207/s15327809jls1501_7

[8] David R Krathwohl. 2002. A revision of Bloom's taxonomy: An overview. *Theory into practice* 41, 4 (2002), 212–218.

[9] Mrityunjay Kumar and Venkatesh Choppella. 2023. A modeling language for novice engineers to design well at SaaS product companies. In *Proceedings of the 16th Innovations in Software Engineering Conference.* 1–5.

[10] George H Mealy. 1955. A method for synthesizing sequential circuits. *The Bell System Technical Journal* 34, 5 (1955), 1045–1079.

[11] M David Merrill. 2002. First principles of instruction. *Educational technology research and development* 50 (2002), 43–59.

[12] Edward F Moore et al. 1956. Gedanken-experiments on sequential machines. *Automata studies* 34 (1956), 129–153.

[13] John Morecroft. 2004. Mental models and learning in system dynamics practice. *Systems modelling: Theory and practice* (2004), 101–126.

[14] Marian Petre. 2013. UML in Practice. In *2013 35th International Conference on Software Engineering (Icse).* IEEE, 722–731.

[15] John D Sterman. 1994. Learning in and about complex systems. *System dynamics review* 10, 2-3 (1994), 291–330.

[16] WeiTek Tsai, XiaoYing Bai, and Yu Huang. 2014. Software-as-a-service (SaaS): perspectives and challenges. *Science China Information Sciences* 57, 5 (2014), 1–15.

[17] L. Von Bertalanffy. 1972. The History and Status of General Systems Theory. *Academy of Management Journal* 15, 4 (Dec. 1972), 407–426. https://doi.org/10.2307/255139

[18] Bernard P Zeigler, Tag Gon Kim, and Herbert Praehofer. 2000. *Theory of modeling and simulation.* Academic press.